CSE Qualifying Exam, Fall 2018 -- 513 Architecture

1. Consider the following loop nest:

```
float x[1024], out[1024], coeff[3];
for (i=0;i<n;i++) {
  out[i] = 0;
  for (j=0;j<3;j++)
    out[i] = out[i] * x[i] + coeff[j];
}
```

a. Is the inner loop parallel? Why or why not?

b. What is the arithmetic intensity of this loop nest?

c. Assume the processor has a peak memory bandwidth of 12.8 GB/s and a peak single precision floating-point throughput of 10 Gflops/s. Is this loop compute bound or memory bound? Why?

d. If the compiler were to unroll the inner loop completely, unroll the outer loop by a factor of 2, and schedule the instructions to hide their latency as much as possible, estimate the minimum number of registers required. Be sure to provide justification.

2. Consider a memory system that has the following characteristics.

memory	type	line size	miss rate	write policy	% dirty	access time	bandwidth to lower level
L1	split	I: 64 bytes D: 16 bytes	I: 2% D: 7%	D: write-back	50%	0	6.4 GB/s
L2	unified	64 bytes	2%	write back	50%	10 ns	30 GB/s to DRAM 300 GB/s to HBM2
DRAM						300 ns	
HBM2						900 ns	

In addition to its normal DRAM (Dynamic RAM, or regular system RAM), this memory system also has a "High Bandwidth Memory 2," in which the software can allocate certain frequently-accessed data structures of its choosing instead of allocating them in DRAM, specifically:

- All program code (instructions) is allocated in DRAM.
- 90% of data accesses are allocated in HBM2.
- 10% of data access are allocated in DRAM.

Assume the base instruction CPI is 1. For a program under test, assume that 15% of all executed instructions are loads and 5% of all executed instructions are stores.

Calculate the overall CPI. State any assumptions.

3. You are trying to decide what type of processor to purchase for a specific application. When running in serial mode, the application spends 90% of its execution time performing operations that it could perform on a vector unit. The application can also run in multi-threaded mode, in which only 1% of the execution time must be serialized as a single thread.

   Processor 1 has four cores and each core has a vector unit that can execute vectorized code with a speedup of 12.

   Processor 2 has eight cores and each core has a vector unit that can execute vectorized code with a speedup of 4.

   Aside from these differences both processors are identical. Which processor do you purchase and why?

Fall 2018 CSE Qualifying Exam
CSCE 531, Compilers

1. **LR-Parsing.** Consider the following augmented grammar G with start symbol S:

$$\begin{array}{lcl} S & \rightarrow & E \\ E & \rightarrow & T \; ? \; E \; : \; E \\ E & \rightarrow & T \\ T & \rightarrow & c \\ T & \rightarrow & (\; E \;) \end{array}$$

(This grammar describes the "if-then-else" operator in C/C++/Java.)

(a) For the grammar G above generate all of the LR(1) sets of items $I_0, I_1, I_2, \ldots, I_{12}$ (thirteen states in all) along with complete transition information for an LALR (LALR(1)) parser. [Note: This is NOT a canonical LR(1) parser; states with common cores are merged.]

(b) Using the sets-of-items constructed in part (b), construct the action table and describe any conflicts. Assume the productions are numbered in order from 0 to 4.

2. **Syntax-Directed Definition: Type Inference with Overloaded Operators.** Consider the grammar for expression syntax, below:

$$
\begin{array}{rcl}
\langle expr \rangle & ::= & \langle expr \rangle \ \textbf{op} \ \langle term \rangle \\
& | & \langle term \rangle \\
\langle term \rangle & ::= & \langle factor \rangle \ \text{':'} \ \textbf{int} \\
& | & \langle factor \rangle \ \text{':'} \ \textbf{real} \\
& | & \langle factor \rangle \\
\langle factor \rangle & ::= & \textbf{var} \\
& | & \text{'('} \ \langle expr \rangle \ \text{')'}
\end{array}
$$

Every expression and subexpression must have one of four possible types: **int**, **real**, **unknown**, or **error**. Give semantic rules to compute a synthesized *type* attribute for each of the three nonterminals according to the following rules:

(a) If any subexpression has type **error**, then the whole expression has type **error**. This rule takes precedence over all the following rules.

(b) If an expression involves the colon (:) operator, then the type of the expression is the same as the type to the right of the colon, *provided* the subexpression to the left is either of the same type or of **unknown** type. Otherwise, the expression has type **error**.

(c) An expression consisting of a single **var** is of **unknown** type.

(d) When **op** is applied to two subexpressions:

i. If either subexpression is of type **real**, then the result is of type **real**.

ii. If both subexpressions are of type **int**, then the result is of type **int**.

iii. Otherwise, the result is of type **unknown**.

(Note the exceptions to this rule given by rule (a), above.)

3. **Control Flow and Liveness Analysis.** The following fragment of 3-address code was produced by a nonoptimizing compiler:

```
 1 start:  i := 0
 2         j := 0
 3 loop:   t1 := a[i]
 4         t2 := b[j]
 5         if t1 < n goto cont
 6         t2 := b[j]
 7         if t2 >= n goto exit
 8 cont:   k := i + j
 9         if t1 >= t2 goto B
10         c[k] := t1
11         t1 := 0
12         i := i + 1
13         goto inc
14 B:      c[k] := t2
15         j := j + 1
16         goto inc
17         k := k + 1
18 inc:    goto loop
19 exit:   no-op
```

Assume that there are no entry points into the code from outside other than at `start`.

(a) (20% credit) Decompose the code into basic blocks $B_1, B_2, \ldots$, giving a range of line numbers for each.

(b) (30% credit) Draw the control flow graph, describe any unreachable code, and coalesce any nodes if possible.

(c) (30% credit) Give a table with 19 rows saying which variables are live immediately before each line number. Assume that `c` and `n` are the only live variables immediately after line 19.

(d) (20% credit) Describe any simplifying transformations that can be performed on the code (i.e., transformations that preserve the semantics but reduce (i) the complexity of an instruction, (ii) the number of instructions, (iii) the number of branches, or (iv) the number of variables).

Algorithms

1. **Find** tight asymptotic bounds on any positive real-valued function $T(n)$ satisfying the following recurrence for all sufficiently large n:

$$T(n) = \frac{1}{4}T\left(\frac{3}{4}n\right) + \frac{3}{4}T\left(\frac{1}{4}n\right) + 1$$

   That is, find an expression $f(n)$, as simple as possible, such that $T(n) = \Theta(f(n))$. Use the substitution method to **prove** that your answer is correct. (Note: Implicit floors or ceilings in the recurrence do not affect the answer.)

2. Suppose you have m nuts and m bolts. Each nut matches exactly one bolt; each bolt matches exactly one nut. However, the sizes are all very similar, so you cannot directly tell whether a given nut is smaller or larger than another nut. Likewise, you cannot directly compare one bolt to another bolt. However, if you select one nut and one bolt, you can determine whether that nut is *too small*, *too big*, or *just right* for that bolt. Your goal is to match each nut with the correct bolt.

   More precisely, you are given a list of nuts $n_1, \ldots, n_m$, and a list of bolts $b_1, \ldots, b_m$. You have access to a subroutine $\textsc{Test}(n_i, b_j)$, which returns -1 if nut n_i is smaller than bolt b_j; $+1$ if nut n_i is larger than bolt b_j; and 0 if nut n_i and bolt b_j match. The output should be a list of ordered pairs $(n_{i_1}, b_{j_1}), \ldots, (n_{i_m}, b_{j_m})$ such that each nut-bolt pair is correctly matched.

   Describe a randomized algorithm for this problem, for which the worst-case expected number of calls to the $\textsc{Test}$ subroutine is $\Theta(m \log m)$. **Explain** why your algorithm works correctly, in enough detail to convince an intelligent but skeptical reader that it is correct.

3. A **palindrome** is a string that is unchanged by reversal. Specifically, a string $S[1, \ldots, n]$ is a palindrome if $S[i] = S[n - i + 1]$ for every $i \in \{1, \ldots, n\}$. Examples: 'hannah', 'smhtiroglalgorithms', and 'x' are palindromes, but 'turtles' is not a palindrome.

   Notice that any string can be partitioned into palindromes performing a series of zero or more 'cuts' that slice it into substrings. The table below shows some examples.

Original String	**Partition into Palindromes**	**Number of Cuts**
abb	a|bb	1
abbc	a|bb|c	2
abba	abba	0
abbab	abba|b	1
abcde	a|b|c|d|e	4

   Describe an algorithm whose input a string of length n, and whose output is the smallest number of cuts needed to partition the input string into palindrome substrings. Your algorithm only needs to compute the *number* of cuts; it does not need to generate locations of those cuts. **Explain** why your algorithm works, in enough detail to convince an intelligent but skeptical reader that it is correct. For full credit, your algorithm should run in $\Theta(n^2)$ time.